CLIPPEDIMAGE= JP407182209A
PUB-NO: JP407182209A
DOCUMENT-IDENTIFIER: JP 07182209 A
TITLE: METHOD AND SYSTEM FOR MONITORING PERFORMANCE OF PROGRAM ON OPERATING
SYSTEM BASE
PUBN-DATE: July 21, 1995
INVENTOR-INFORMATION:
NAME
BOLOSKY, WILLIAM J
RASHID, RICHARD F
INT-CL_(IPC): G06F011/34; G06F009/06
ABSTRACT:
PURPOSE: To monitor the performance of the kernel of an operating system and that of a user level program.

CONSTITUTION: An operating system 28 is provided with facilities in its kernel to monitor the performance of a program. These facilities monitor not only a part of the operating system like the kernel 30 but also user level programs. Facilities counts instructions and/or function calls to provide a performance reference convenient for the user of a system. This counted value is sent to a user level monitor program. Since facilities are included in the kernel 30, directly monitored by facilities.

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（A）　(11)特許出願公開番号

特開平7－182209

(43)公開日　平成７年(1995)７月21日

(51)Int.Cl.[6]	識別記号	庁内整理番号	Ｆ Ｉ	技術表示箇所
Ｇ０６Ｆ　11/34		Ｎ　9290－5B		
9/06	５４０	Ｕ　9367－5B		

審査請求　未請求　請求項の数28　ＯＬ　（全８頁）

(21)出願番号　特願平6－268813

(22)出願日　平成６年(1994)11月１日

(31)優先権主張番号　０８／１４７６４５

(32)優先日　1993年11月４日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　391055933
マイクロソフト　コーポレイション
ＭＩＣＲＯＳＯＦＴ　ＣＯＲＰＯＲＡＴＩＯＮ
アメリカ合衆国　ワシントン州　98052－6399　レッドモンド　ワン　マイクロソフト　ウェイ　（番地なし）

(72)発明者　ウィリアム　ジェイ　ボロスキー
アメリカ合衆国　ワシントン州　98027　イサカ　サウスイースト　ミロールモント　ドライヴ　24622

(74)代理人　弁理士　中村　稔　（外６名）

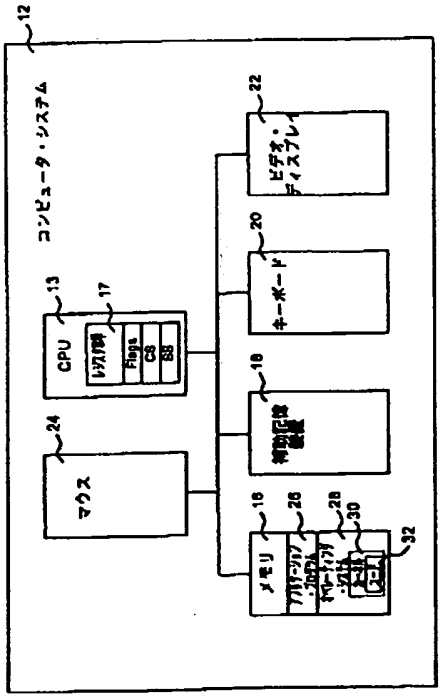
最終頁に続く

(54)【発明の名称】　オペレーティング・システム・ベースのプログラムの性能モニタ方法およびシステム

(57)【要約】

【目的】　オペレーティング・システムのカーネルの性能およびユーザ・レベル・プログラムの性能をモニタできるようにする。

【構成】　オペレーティング・システムは、プログラムの性能をモニタするために、そのカーネル内にファシリティを備えている。このファシリティは、カーネルのようなオペレーティング・システムの部分だけでなく、ユーザ・レベル・プログラムもモニタする。ファシリティは、システムのユーザに便利な性能基準を提供するために、命令および関数コールのカウントの双方またはいずれか一方を行う。このカウント値は、ユーザ・レベル・モニタ・プログラムに送られる。カーネルにファシリティを含めることにより、性能のモニタの速度を向上させることができ、オペレーティング・システムをファシリティによって直接モニタすることもできる。



1

【特許請求の範囲】

【請求項１】　命令を実行するときにシステム・レベルまたはユーザ・レベルで実行するプロセッサと、カーネルを含むオペレーティング・システムを記憶するための記憶機構とを備えたデータ処理システムにおける方法であって、

前記プロセッサによって実行される命令をカウントするための、前記オペレーティング・システムのカーネルのファシリティ、および前記ファシリティによってカウントされる命令をモニタするためのユーザ・レベル・モニタリング・プログラムを提供し、

前記プロセッサ上でコードの一部を実行し、

前記オペレーティング・システムのカーネルの前記ファシリティを使用して、前記プロセッサ上で前記コードの一部を実行するときに実行される命令数をカウントし、ならびに前記ユーザ・レベル・モニタリング・プログラムに、実行された命令数のカウントを報告するステップ、

を含む方法。

【請求項２】　前記プロセッサ上で実行されるコードの一部が、オペレーティング・システムのカーネルの一部である、請求項１に記載の方法。

【請求項３】　前記プロセッサ上で実行されるコードの一部が、ユーザ・レベル・プログラムの一部である、請求項１に記載の方法。

【請求項４】　前記プログラム上で実行されるコードの一部が、オペレーティング・システムのカーネルの部分でない、請求項１に記載の方法。

【請求項５】　前記プロセッサ上で前記コードの一部を実行するときに実行される命令数のカウントを前記記憶機構に記憶するステップ、含む請求項１に記載の方法。

【請求項６】　前記コードの一部が関数であり、前記コードの一部を前記プロセッサ上で実行するステップが、前記関数を前記プロセッサ上で実行するステップを含むものである、請求項１に記載の方法。

【請求項７】　前記プロセッサ上で実行されるコードの一部をチューニングし、前記コードの一部を前記プロセッサ上で実行するときに実行される命令数を減少させるステップ、

を含む請求項１に記載の方法。

【請求項８】　前記データ処理システムが、第２のプロセッサを含む分散システムであり、

第２のコードの一部を前記第２のプロセッサ上で実行し、

前記オペレーティング・システムのカーネルのファシリティを使用して、前記第２のプロセッサ上で前記第２のコードの一部を実行するときに実行される命令数をカウントし、および前記第２のコードの一部を実行するときに実行される命令の前記カウントを、前記ユーザ・レベル・モニタリング・プログラムに報告するステップ、

2

を含む請求項１に記載の方法。

【請求項９】　前記第２のプロセッサ上で前記第２のコードの一部を実行するときに実行される命令数の前記カウントを、前記記憶機構に記憶するステップ、

を含む請求項８に記載の方法。

【請求項１０】　命令を実行するときにシステム・レベルまたはユーザ・レベルで実行するプロセッサ、およびカーネルを含むオペレーティング・システムを記憶するための記憶機構を備えたデータ処理システムにおいて、プログラムの実行中にコードの一部が呼び出された回数をカウントするための、前記オペレーティング・システムのカーネルのファシリティ、および前記ファシリティによってカウントされる命令をモニタするためのユーザ・レベル・モニタリング・プログラムを提供し、

第１のコードの一部を含むプログラムの少なくとも一部を前記プロセッサ上で実行し、

前記オペレーティング・システムのカーネルの前記ファシリティを使用して、前記第１のコードの一部が、前記プロセッサ上で前記プログラムの一部の実行中に呼び出される回数をカウントし、ならびに前記ユーザ・レベル・モニタリング・プログラムに前記カウントを報告するステップ、

を含む方法。

【請求項１１】　前記プログラムが、前記オペレーティング・システムのカーネルの一部である、

請求項10に記載の方法。

【請求項１２】　前記プログラムがユーザ・レベル・プログラムである、請求項10に記載の方法。

【請求項１３】　前記プログラムが、前記オペレーティング・システムのカーネルの一部でない、

請求項10に記載の方法。

【請求項１４】　前記第１のコードの一部が前記プログラムの一部の実行中に呼び出される回数のカウントを、前記記憶機構に記憶するステップ、

を含む請求項10に記載の方法。

【請求項１５】　前記第１のコードの一部が、関数であり、

前記オペレーティング・システムのカーネルの前記ファシリティを使用して、前記第１のコードの一部が前記プロセッサ上での前記プログラムの一部の実行中に呼び出される回数をカウントする前記ステップが、前記オペレーティング・システムのカーネルの前記ファシリティを使用して、前記関数が前記プログラムの一部の実行中に呼び出される回数をカウントするステップを含むものである、

請求項10に記載の方法。

【請求項１６】　前記第１のコードの一部が、前記プログラムの一部の実行中に呼び出される回数のカウントを使用して、前記プログラムの性能のチューニングを指標するステップ、

を含む請求項10に記載の方法。

【請求項１７】　前記プログラムが、第２のコードの一部を含み、

前記オペレーティング・システムのカーネルの前記ファシリティを使用して、前記第２のコードの一部が、前記プロセッサ上での前記プログラムの一部の実行中に呼び出される回数をカウントし、および前記第２のコードの一部が呼び出された回数のカウントを、前記ユーザ・レベル・モニタリング・プログラムに報告するステップ、

を含む請求項10に記載の方法。

【請求項１８】　前記第２のコードの一部が前記プロセッサ上での前記プログラムの一部の実行中に呼び出される回数のカウントを、前記記憶機構に記憶するステップ、

を含む請求項17に記載の方法。

【請求項１９】　前記データ処理システムが、第２のプロセッサを含む分散システムであり、

コードの一部を含む第２のプログラムの少なくとも一部を、前記プロセッサ上で実行し、

前記オペレーティング・システムのカーネルの前記ファシリティを使用して、前記第２のプログラムの前記コードの一部が前記第２のプログラムの一部の実行中に呼び出される回数をカウントし、および前記第２のプログラムの前記コードの一部が呼び出された回数のカウントを、前記ユーザ・レベル・モニタリング・プログラムに報告するステップ、

を含む請求項10に記載の方法。

【請求項２０】　前記第２のプログラムの前記コードの一部が前記第２のプログラムの一部の実行中に呼び出される回数のカウントを、前記記憶機構に記憶するステップ、

を含む請求項19に記載の方法。

【請求項２１】　命令を実行するためのシステム・レベルまたはユーザ・レベルで実行可能なプロセッサ、およびカーネルを有するオペレーティング・システムを記憶するための記憶機構を備えたデータ処理システムにおいて、

前記プロセッサによって実行される命令をカウントするためのファシリティを、前記オペレーティング・システムのカーネルに提供し、

前記プログラムの他の関数を呼び出す第１の関数を前記プロセッサ上で実行し、

前記オペレーティング・システムのカーネルの前記ファシリティを使用して、前記第１の関数が前記プログラムの実行中に呼び出されたときに、前記第１の関数および前記他の関数で実行される命令の関数によって分類される個別のカウントを求め、ならびに前記ユーザ・レベル・モニタリング・プログラムに前記カウントを報告するステップ、

を含む方法。

【請求項２２】　前記プログラムが、前記オペレーティング・システムのカーネルの一部である、

請求項21に記載の方法。

【請求項２３】　前記プログラムがユーザ・レベル・プログラムである、

請求項21に記載の方法。

【請求項２４】　前記プログラムが、前記オペレーティング・システムのカーネルの一部でない、

請求項21に記載の方法。

【請求項２５】　前記個別のカウントを前記記憶機構に記憶するステップを含む、

請求項21に記載の方法。

【請求項２６】　前記プログラムの性能のチューニングを指標する前記個別のカウントを使用して、前記プログラムの性能をチューニングするステップ、

を含む請求項21に記載の方法。

【請求項２７】　命令からなるモニタの対象プログラムの性能を測定するシステムであって、

ユーザ・レベル・モニタリング・プログラム、および実行される命令をカウントするファシリティを含むカーネルを備えたオペレーティング・システムを記憶するための記憶機構、ならびに(i) 前記モニタの対象プログラムと前記ユーザ・レベル・モニタリング・プログラムを実行するためのプログラム実行ユニット、(ii)前記ファシリティを呼び出して、前記モニタの対象プログラムの少なくとも一部によって実行される命令数をカウントするファシリティ・インボーカ、および(iii) 前記ファシリティによって収集されたカウントを、前記ユーザ・レベル・モニタリング・プログラムに報告するカウント・レポータを備えたプロセッサ、を含むシステム。

【請求項２８】　関数を含み、命令からなるモニタ対象のプログラムの性能を測定するシステムであって、ユーザ・レベル・モニタリング・プログラム、および関数が呼び出される回数をカウントするファシリティを含むカーネルを備えたオペレーティング・システムを記憶するための記憶機構、ならびに(i) 前記モニタの対象プログラムと前記ユーザ・レベル・モニタリング・プログラムを実行するためのプログラム実行ユニット、(ii)前記ファシリティを呼び出して、前記モニタの対象プログラムの実行中に、前記モニタの対象プログラム内の前記関数が呼び出される回数をカウントするファシリティ・インボーカ、および(iii) 前記カウントを、前記ユーザ・レベル・モニタリング・プログラムに報告するカウント・レポータを備えたプロセッサ、を含むシステム。

【発明の詳細な説明】

【０００１】

【産業上の利用分野】この発明は、データ処理システムに関し、具体的には、データ処理システム上で実行されるプログラムの性能（パフォーマンス）をモニタ（モニタリング）する方法およびシステムに関する。

5

【0002】

【従来の技術】データ処理システムで実行されるプログラムの性能（パフォーマンス）をモニタするための様々な統計的手法が、これまでに開発されてきている。最も著名な手法の一つとして、プログラムの各関数の実行時間をモニタするものがある。このアプローチによると、多くの場合に、関数の処理時間の計測に使用されるクロックの周期が大きすぎるために、処理時間が各関数にどのように分配されているかの正確な状況を把握できない点に難点がある。プログラムの性能をモニタするもう一つの統計的手法として、命令をカウントするものがある。命令をカウントする手法は、一般に、アプリケーション・プログラムのようなユーザ・レベルのプログラム（システム・レベルのプログラムに対するものとして）の性能をモニタするのに限定されている。

【0003】

【発明が解決しようとする課題】これまでに、これらの手法を、オペレーティング・システムのカーネルの性能をモニタするのに適用することはできなかった。このような制限により、プログラマが、プログラムの性能を向上させるために統計的手法を利用できないことが多かった。

【0004】

【課題を解決するための手段】命令を実行するプロセッサ、ならびにオペレーティング・システムおよびユーザ・レベルのモニタリング・プログラムを記憶するための記憶機構を備えているデータ処理システムにおいて、この発明による方法が実行される。プロセッサは、ユーザ・レベルおよびシステム・レベルの命令を実行する。データ処理システムは、分散システムであってもよい。

【0005】この方法において、ファシリティが、プログラムの性能をモニタするために、オペレーティング・システムのカーネルに備えられる。例えば、このファシリティは、プロセッサによって実行される命令をカウントするものであってもよいし、コードの一部がプログラムの実行中に呼び出された回数をカウントするものであってもよい。また、ファシリティは、各関数で実行される命令数を示す関数によって分類された多くの個別のカウントを提供するものであってもよい。プログラムのコードの一部が、プロセッサによって実行される。このとき、オペレーティング・システムのカーネルにあるファシリティは、システムの性能をモニタするために使用される。

【0006】使用されるファシリティのタイプは、そのファシリティによってカウントされる値を決定する。カウントは、ユーザ・レベル・モニタリング・プログラムに報告される。例えば、ファシリティは、プログラムのコードの一部を実行したときの命令数をカウントするものであってもよいし、プログラムのコードの一部が呼び出される回数をカウントするものであってもよい。また、ファシリティは、第1の関数がプログラムの実行中に呼び出されるときに、関数において実行される命令を備えた関数によって分類された個別のカウントを提供するものであってもよい。

6

【0007】プログラムは、ユーザ・レベル・プログラムであってよいし、オペレーティング・システムのカーネルの一部であってもよい。

【0008】

【実施例】この発明の好ましい実施例は、プログラムの性能をモニタ（モニタリング）するファシリティを、オペレーティング・システムのカーネルに備えている。このファシリティは、多くのオプションをユーザに提供する。第1に、このファシリティにより、ユーザは、ある関数が呼び出されるごとに、その関数によって実行される命令数をカウントすることができる。第2に、このファシリティにより、ユーザは、ある特定の関数への呼び出し回数および頻度を、その関数において実行される命令をカウントすることなく、カウントすることができる。第3に、このファシリティにより、ユーザは、初期関数の呼び出しの結果、呼び出される各関数において実行される命令数をカウントすることができる。このファシリティは、オペレーティング・システムのカーネルに設けられるので、アプリケーション・プログラムのようなユーザ・レベル・プログラムの性能をモニタすることに使用できるだけでなく、カーネルの性能およびオペレーティング・システムの他の部分の性能をモニタすることに使用することもできる。ファシリティをオペレーティング・システムのカーネルに設けることにより、モニタの速度を向上させることもでき、モニタの対象となるプログラムに変更を加えることなく、性能のモニタを行う簡単なアプローチを提供することができる。

【0009】図1に示すように、この実施例を、分散システム10において実現することができる。この発明を、単一のプロセッサ・システムにおいて実行できることも理解できるであろう。この分散システム10は、ネットワーク14を介して通信を行う多くのコンピュータ・システム12を含んでいる。各コンピュータ・システム12は、実行しているプログラムのトレースを同時に行うことができる。ネットワーク14は、ローカル・エリア・ネットワーク（LAN）やワイド・エリア・ネットワークを含む多くの異なるタイプのネットワークのいずれであってもよい。この技術分野の専門家ならば、分散システム10が、図1に示すものとは異なる個数のコンピュータ・システムを含むこともできることを容易に理解するであろう。この分散システムにおいては、4台のコンピュータ・システムが示されているにすぎない。

【0010】図2は、コンピュータ・システム12の一つに含まれる構成要素を詳細に示すブロック図である。各コンピュータ・システムが、このシステム構成を有する必要は必ずしもなく、ここでは、このシステム構成が、

単に示されているにすぎない。コンピュータ・システム12は、中央処理装置（ＣＰＵ）13およびメモリ16を備えている。ＣＰＵ13は、複数のレジスタ17を備えている。レジスタ17には、ＦＬＡＧレジスタおよびＳＳ（Stack Segment ：スタック・セグメント）レジスタが含まれている。これらのレジスタについては、後に詳述する。メモリ16は、ワシントンのレドモンドにあるマイクロソフト社（Microsoft Corporation ）により販売されている、マイクロソフトＮＴオペレーティング・システムのような分散オペレーティング・システム28のコピーを保持している。この実施例においては、各コンピュータ・システム12が、オペレーティング・システムのコピーを保持し、実行する。オペレーティング・システム28は、カーネル30を含んでいる。カーネル30には、性能をモニタすることをサポートするコード部32がある。コード部32については、後に詳述する。メモリ16には、ＣＰＵ13によって実行される少なくとも一つのアプリケーション・プログラム26も記憶されている。しかし、このアプリケーション・プログラム26は、メモリ16に記憶されている必要は必ずしもない。

【００１１】また、コンピュータ・システム12は、多くの周辺装置を備えている。これらの周辺装置には、補助記憶装置（例えば、磁気ディスク・ドライブ）18、キーボード20、ビデオ・ディスプレイ22およびマウス24が含まれる。

【００１２】図３は、コード部32を詳細に示すブロック図である。ＡＰＩは、ブレークポイント・ファシリティ34を含んでいる。このブレークポイント・ファシリティ34は、ＣＰＵ13で実行されるプログラム内のブレークポイントの生成をサポートする。また、コード部32は、シングル・ステップの割込みを制御するシングル・ステップ割込ハンドラ36を含んでいる。ブレークポイント・ファシリティ34およびシングル・ステップ割込ハンドラ36によって提供される機能を説明するために、まず、ブレークポイントとは何か、シングル・ステップ割込みとは何かを知ることが有益である。

【００１３】シングル・ステップ割込みとは、プログラムを１ステップごとに処理（シングル・ステッピング）するときに使用される割込みをである。シングル・ステッピングは、ＣＰＵ13の一つの処理モードであり、管理プログラムの制御の下、一時に一つの命令を実行するモードである。シングル・ステッピングにより、デバッガまたは性能モニタは、プログラムをゆっくりと実行することができ、プログラムの実行を注意深くモニタすることができる。ここでは、シングル・ステップ割込ハンドラ36は、モニタされているプログラムが一時に一つの命令を実行する管理プログラムである。一般に、一つの命令が実行されると、単一のステップ割込みが発生し、プロセッサの制御が管理プログラムに切り換えられる。80X86 （80186 、80286 等）マイクロプロセッサのような

多くのマイクロプロセッサは、マイクロプロセッサをシングル・ステッピング・モードに切り換える便利な機構を備えている。80X86 マイクロプロセッサには、シングル・ステップ・トラップ・フラグが、ＦＬＡＧレジスタ（図２）に備えられている。このシングル・ステップ・トラップ・フラグの値が１の場合には、マイクロプロセッサは、シングル・ステップ・モードを実行し、一方、シングル・ステップ・トラップ・フラグの値が０の場合には、マイクロプロセッサは、シングル・ステップ・モードを実行しない。

【００１４】ブレークポイントとは、プログラムの自由な実行をブレークするために使用され、プログラムのデバッグおよびプログラムの性能のモニタに有効なブレークポイント割込みである。シングル・ステップ・モードにおいては、プログラムがゆっくりと実行されるので、プログラムを全て実行することは難しい。多くの場合に、プログラムの大部分は既にデバッグまたはチューンされており、わずかな部分にのみ、詳細な試験が必要とされる。ブレークポイント割込みは、このような状況において特に適している。ブレークポイント割込みは、プログラム中に特別のオペコード（演算コード：opcode）を挿入することにより、発生する。このオペコードは、実行時に、ブレークポイント割込みを発生させる。ブレークポイント割込みが発生すると、制御は、ブレークポイント・ファシリティ34に渡される。ブレークポイント・ファシリティ34は、さらに必要となる処理を行うこともある。この実施例におけるブレークポイントおよびシングル・ステップ割込みの機能については、以下に述べる図４および図５の説明において詳述する。

【００１５】ブレークポイント・ファシリティ34およびシングル・ステップ・ハンドラ36に加えて、コード部32は、文脈スイッチに関係したコード38を含んでいる。このコード38は、文脈スイッチ用のフックを備え、文脈スイッチを検査して、新しい文脈をシングル・ステップとすべきかどうかを決定するハンドラを含んでいる。新しい文脈がシングル・ステップとされるならば、このハンドラは、シングル・ステッピングを確実に発生させる。

【００１６】コード部32は、カウンタ値40も含んでいる。上述したように、この実施例では、ユーザは、関数において実行される命令数、関数がプログラム内で呼び出される回数および初期関数を呼び出した結果として呼び出される各関数用に実行される命令数をモニタすることができる。これらの各カウントは、個別のカウンタに記憶され、そのカウンタの値が、コード部32のデータ・エリアのカウンタ値40として保持される。

【００１７】図４は、この実施例においてプログラムまたはプログラムの一部の性能をモニタするために実行される処理の流れを示すフローチャートである。まず、ユーザは、性能基準（性能メトリック）を選択する（ステップ42）。続いて、ダイアログ・ボックスのメニューの

ようなユーザ・インタフェースが、ユーザが使用しているコンピュータ・システム12（図１）のビデオ・ディスプレイ22（図２）に表示される。ユーザは、このユーザ・インタフェースを使用することにより、所望の性能基準を選択することができる。上述したように、この実施例において求められる多くの異なる性能基準がある。また、ユーザは、プログラム中のどの関数をモニタするかを選択する（ステップ44）。性能基準が選択され、モニタされる関数が選択されると、ブレークポイント割込みを生成するオペコードがモニタされる関数のそれぞれに挿入され、プログラムまたはプログラムの一部が実行される（ステップ46）。適切なカウンタ値40が、プログラムの実行に伴い計算される。プログラムの実行に伴い、カウンタ値40は、ビデオ・ディスプレイ22に表示される（ステップ48）。プログラムまたはプログラムの一部の実行が終了すると、ユーザは、プログラムを適切にチューニング（チューン）するための指標となる性能データを検討することができる（ステップ50）。性能データをどのようにプログラムのチューニングに使用するかは、収集されたデータのタイプに依存する。この技術の専門家には、プログラムを適切にチューニングするための性能基準データを使用方法が分かるであろう。

【００１８】図５は、関数ごとに行われるブレークポイントの挿入およびプログラムの実行（すなわち、図４のステップ46）の詳細を示すフローチャートである。上述したように、ブレークポイントのオペコードが、モニタされる各関数の最初の命令として置かれる（ステップ52）。続いて、システムは、関数の実行を開始する（ステップ54）。ブレーク・ポイントは、関数の最初の命令として置かれているので、カーネルは、ブレークポイントを発見し（ステップ56）、ユーザがその関数用に要求した性能基準がどれかを決定する（ステップ58）。要求された性能基準が関数の呼び出し回数のカウントだけならば、その関数の呼び出し回数を示すために、カウンタ値40（図３）のうちの、その関数の呼び出し回数をカウントするカウンタの値がインクリメントされる（ステップ60）。このカウンタがインクリメントされると、その関数は、再スタートし、フルスピードで実行を再開する。

【００１９】この実施例において実行される処理について説明する前に、ユーザが、実行される命令数をモニタしたいときに、「スレッド」および「スタック」を紹介しておくことが有益である。一つのスレッドは、他のスレッドと独立に、かつ、同時に実行できる仕事の単位である。オペレーティング・システム28は、マルチスレッド・システムである。この技術分野の専門家ならば、必ずしもマルチスレッド・システムで実行される必要がないことを理解するであろう。また、シングル・プロセスおよびマルチプロセシング・システムで実行することもできる。マルチスレッド・オペレーティング・システム

は、複数のスレッドを同時に実行することができる。スタックとは、後入れ先出し（ラスト・イン・ファースト・アウト（Last-In-First-Out ：LIFO））のデータ構造を持つものであり、スレッド用のデータおよびレジスタ情報を記憶するために使用される。各スレッドは、そのスレッド専用のスタック・セグメント（Stack Segment ：ＳＳ）部を有する。ＳＳレジスタ（図２）は、実行されているプログラムのスレッドのスタックを記憶するメモリのセグメントをポイントする。各スレッドは、そのスタック（すなわち、そのスレッド専用のスタック・セグメント部）の先頭をポイントする、そのスレッドに関連したスタック・ポインタ（Stack Pointer ：ＳＰ）の値を有する。この技術分野の専門家ならば、スタックを使用しない環境においても、この発明を実行できることを理解するであろう。

【００２０】図５に戻って、ステップ58において、ユーザが、関数の呼び出し回数のカウントではなく、実行される命令数のカウントを望んでいると判定されると、現在実行中のスレッドのスタック・ポインタがメモリ16に記憶される（ステップ64）。ＦＬＡＧレジスタ（図２）のシングル・ステップ・トラップ・フラグは、ＣＰＵ13がシングル・ステッピング・モードになるようにセットされる（ステップ66）。シングル・ステッピング・モードでは、命令が実行されるごとに、現在実行されている関数の命令数を追跡するカウンタ値が、インクリメントされる（ステップ68）。これらの命令は、スレッドがモニタされている関数を抜け出すか、スレッドが終了するまで、カウントされる。スレッドがそれ自身をスケジュールしないと、カウントしている命令は中断されるかも知れないが、スレッドが再びスケジュールされた後は、命令のカウントが再開される。

【００２１】この発明について実施例を参照しながら説明したが、この技術分野の専門家ならば、特許請求の範囲において定められるこの発明から逸脱することなく、様々な変更や具体化を行うことができる。

【図面の簡単な説明】

【図１】この発明の好ましい実施例を実行する分散システムの構成を示すブロック図である。

【図２】図１に示すコンピュータ・システムの構成を詳細に示すブロック図である。

【図３】図２におけるＡＰＩを詳細に示すブロック図である。

【図４】この発明の好ましい実施例におけるプログラムの性能のモニタおよびプログラムをチューニングの処理の流れを、ユーザの操作に基づいて示すフローチャートである。

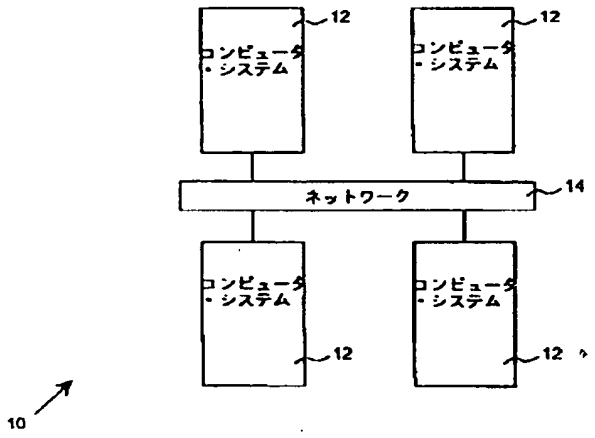
【図５】この発明の好ましい実施例においてモニタが行われるときに、ブレークポイントがどのように挿入されるか、およびプログラムがどのように実行されるかを詳細に示すフローチャートである。

11　　12

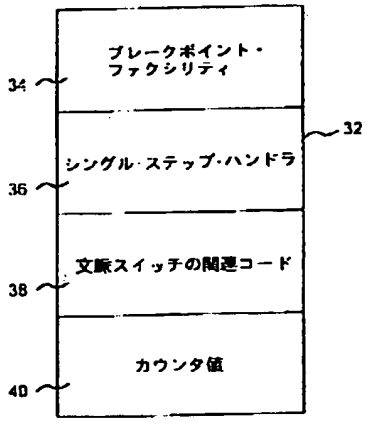
【符号の説明】

10　分散システム
12　コンピュータ・システム
13　ＣＰＵ
14　ネットワーク
16　メモリ
18　補助記憶装置
20　キーボード
22　ビデオ・ディスプレイ
26　アプリケーション・プログラム
28　オペレーティング・システム
30　カーネル
32　コード部

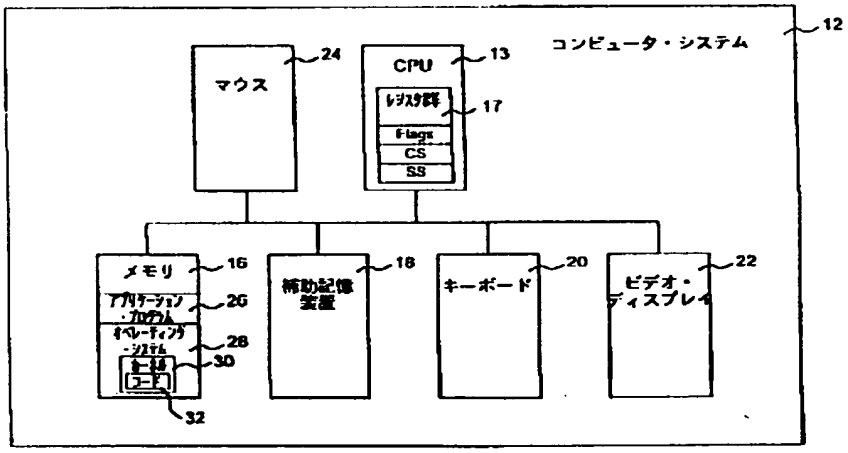
【図１】



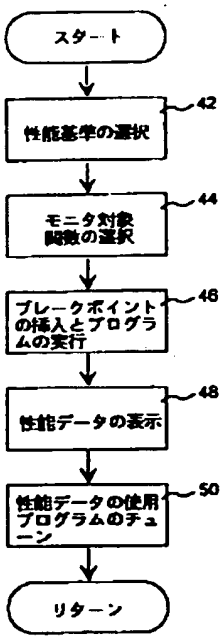
【図３】



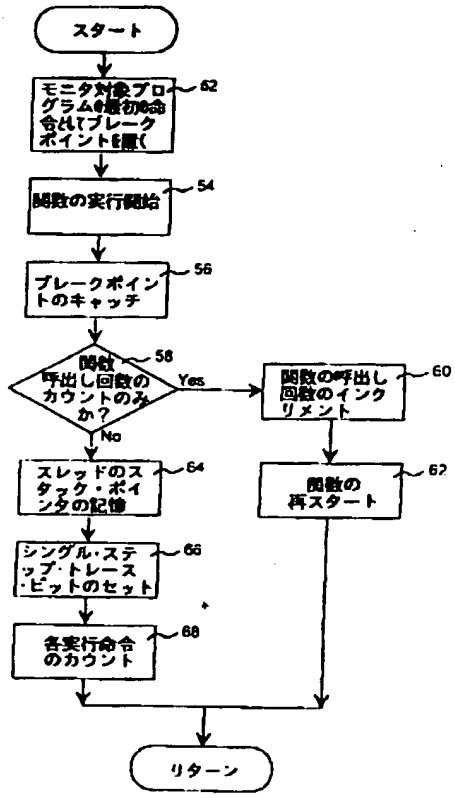
【図２】



【図４】



【図5】



フロントページの続き

(72)発明者　リチャード　エフ　ラシッド
アメリカ合衆国　ワシントン州　98072
ウッディンヴィル　ノースイースト　ワン
ハンドレッドアンドサーティサード　スト
リート　18601